QT

Team Emertxe

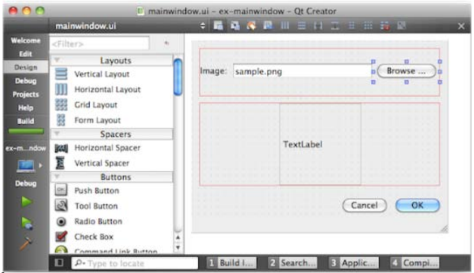
Qt Designer

Dialogs
and Designer

- Dialogs
- Common Dialogs
- Qt Designer

Qt Designer

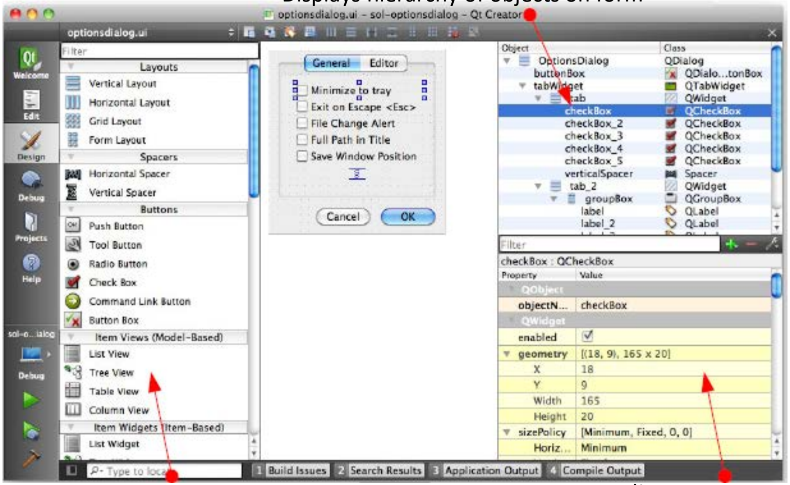
- Design UI forms visually
- Visual Editor for
  - Signal/slot connections
  - Actions
  - Tab handling
  - Buddy widgets
  - Widget properties
  - Integration of custom widgets
  - Resource files



Designer Views

Object Inspector

Displays hierarchy of objects on form



Object	Class
OptionsDialog	QDialog
buttonBox	QDialo...tonBox
tabWidget	QTabWidget
tab	QWidget
checkBox	QCheckBox
checkBox_2	QCheckBox
checkBox_3	QCheckBox
checkBox_4	QCheckBox
checkBox_5	QCheckBox
verticalSpacer	Spacer
tab_2	QWidget
groupBox	QGroupBox
label	QLabel
label_2	QLabel

Filter

checkBox : QCheckBox

Property	Value
QObject	
objectN...	checkBox
QWidget	
enabled	✓
geometry	[(18, 9), 165 x 20]
X	18
Y	9
Width	165
Height	20
sizePolicy	[Minimum, Fixed, 0, 0]
Horiz...	Minimum

Widget Box

Provides selection of widgets, layouts

Property Editor

Displays properties of selected object

Editing Modes

- Widget Editing
  - Change appearance of form
  - Add layouts
  - Edit properties of widgets
- Signal and Slots Editing
  - Connect widgets together with signals & slots
- Buddy Editing
  - Assign buddy widgets to label
  - *Buddy widgets help keyboard focus handling correctly*
- Tab Order Editing
  - Set order for widgets to receive the keyboard focus

UI Form Files

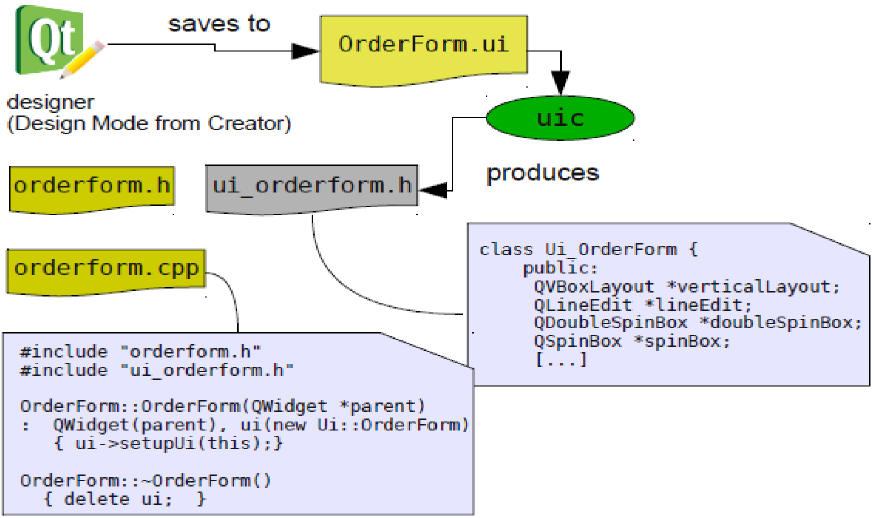
- Form stored in .ui file
  - format is XML
- uic tool generates code
  - From myform.ui
    - to ui_myform.h

```
// ui_mainwindow.h
class Ui_MainWindow {
public:
QLineEdit *fileName;
... // simplified code
void setupUi(QWidget *) { /* setup widgets */ }
};
```

- Form ui file in project (.pro)

  FORMS += mainwindow.ui

From .ui to C++



```
class Ui_OrderForm {
    public:
     QVBoxLayout *verticalLayout;
     QLineEdit *lineEdit;
     QDoubleSpinBox *doubleSpinBox;
     QSpinBox *spinBox;
     [...]
```

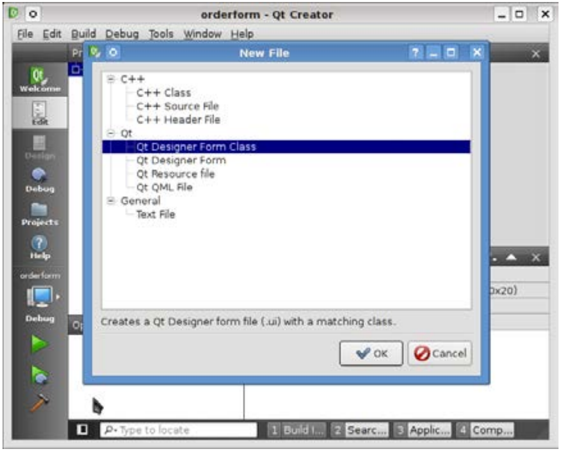
```
#include "orderform.h"
#include "ui_orderform.h"

OrderForm::OrderForm(QWidget *parent)
:   QWidget(parent), ui(new Ui::OrderForm)
    { ui->setupUi(this);}

OrderForm::~OrderForm()
  { delete ui;  }
```

Form Wizards

- **Add New…** "Designer Form"
  - or "Designer Form Class" (for C++ integration)



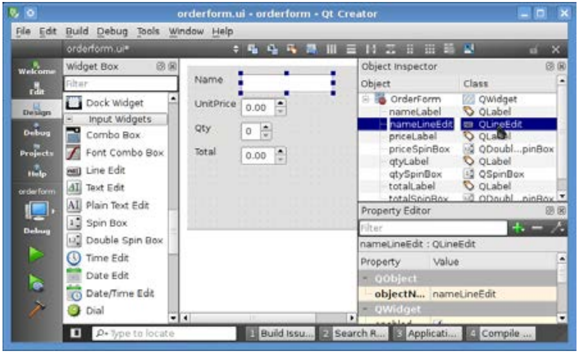
C++
C++ Class
C++ Source File
C++ Header File
Qt
Qt Designer Form Class
Qt Designer Form
Qt Resource file
Qt QML File
General
Text File

Creates a Qt Designer form file (.ui) with a matching class.

ΣMERTXE

Naming Widgets

1. Place widgets on form
2. Edit objectName property



Object	Class
OrderForm	QWidget
nameLabel	QLabel
nameLineEdit	QLineEdit
priceLabel	QLabel
priceSpinBox	QDoubl...pinBox
qtyLabel	QLabel
qtySpinBox	QSpinBox
totalLabel	QLabel
totalSpinBox	QDoubl...pinBox

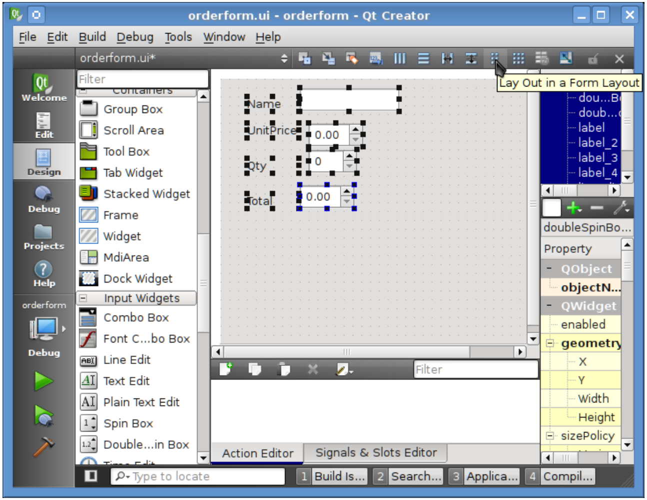
nameLineEdit : QLineEdit

Property	Value
QObject	
objectN...	nameLineEdit
QWidget	

- *objectName defines member name in generated code*

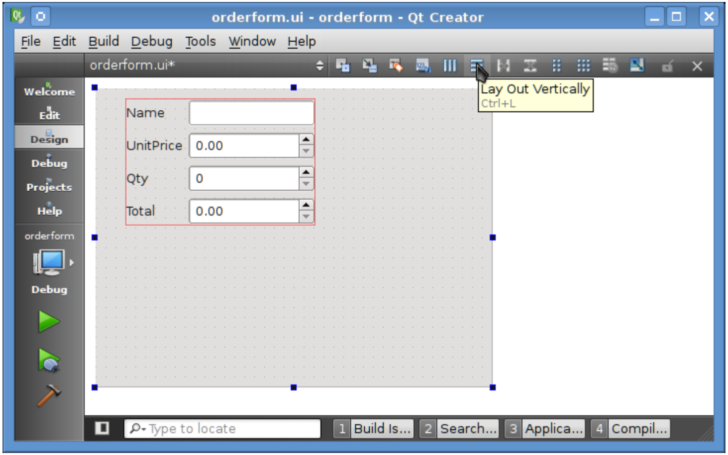
Form layout

`QFormLayout`: Suitable for most input forms



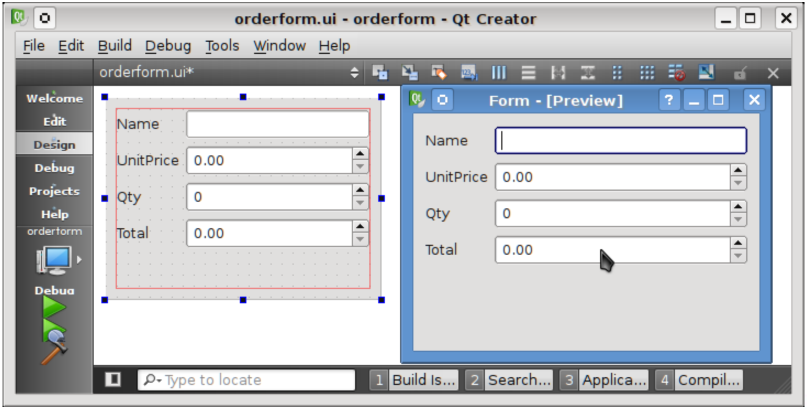
Top-Level Layout

- First layout child widgets
- Finally select empty space and set top-level layout



Preview Mode

Check that widget is nicely resizable



Name	
UnitPrice	0.00
Qty	0
Total	0.00

Code Integration

```
// orderform.h
class Ui_OrderForm;
class OrderForm : public QDialog {
private:
Ui_OrderForm *ui; // pointer to UI object
};
```

- "Your Widget" derives from appropriate base class
- ***ui** member encapsulate UI class
- Makes header independent of designer generated code

Code Integration

```
// orderform.cpp
#include "ui_orderform.h"
OrderForm::OrderForm(QWidget *parent)
: QDialog(parent), ui(new Ui_OrderForm) {
ui->setupUi(this);
}
OrderForm::~OrderForm() {
delete ui; ui=0;
}
```

- *Default behavior in Qt Creator*

Signals and Slots

- Widgets are available as public members
  - ui->fileName->setText("image.png")
    - *Name based on widgets object name*
- You can set up signals & slots traditionally…
  - connect(ui->okButton, SIGNAL(clicked()), …
- Auto-connection facility for custom slots
  - Automatically connect signals to slots in your code
    - Based on object name and signal
  - void on_*objectName_signal*(*parameters*);
    - Example: *on_okButton_clicked()* slot
  - Automatic connections
- Qt Creator: right-click on widget and "Go To Slot"
  - Generates a slot using auto-connected name

Loading .ui files

- Forms can be processed at runtime
  - Produces dynamically generated user interfaces
- Disadvantages
  - Slower, harder to maintain
  - Risk: .ui file not available at runtime
- Loading .ui file

```
QUiLoader loader;
QFile file("forms/textfinder.ui");
file.open(QFile::ReadOnly);
QWidget *formWidget = loader.load(&file, this);
```

- Locate objects in form

```
ui_okButton = qFindChild<QPushButton*>(this,
"okButton");
```

Hands-on

- Lab 10: Order form
  - Objectives
  - Template code

Model/View modules

Objectives

- Model/View Concept
- Custom Models
- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Objectives

Using Model/View

- Introducing to the concepts of model-view
- Showing Data using standard item models

Custom Models

- Writing a simple read-only custom model.
- Editable Models
- Custom Delegates
- Using Data Widget Mapper
- Custom Proxy Models
- Drag and Drop

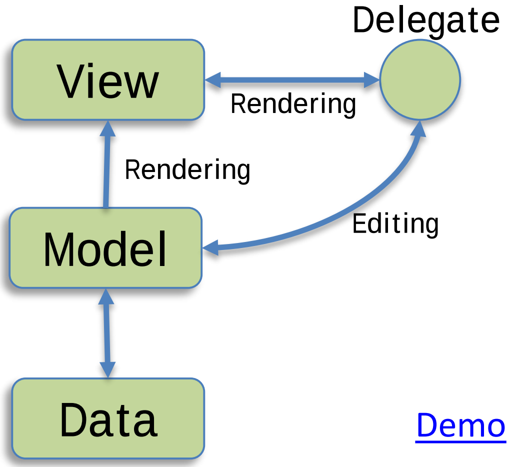
Model/View

- Model/View Concept
- Custom Models
- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Why Model/View?

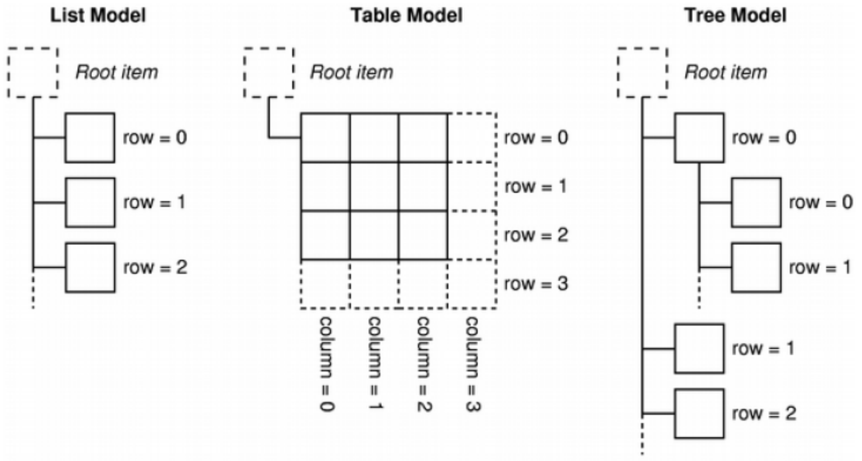
- **Isolated domain-logic**
  - From input and presentation
- **Makes Components Independent**
  - For Development
  - For Testing
  - For Maintenance
- **Foster Component Reuse**
  - Reuse of Presentation Logic
  - Reuse of Domain Model

Model / View
Components



Demo

Model Structures



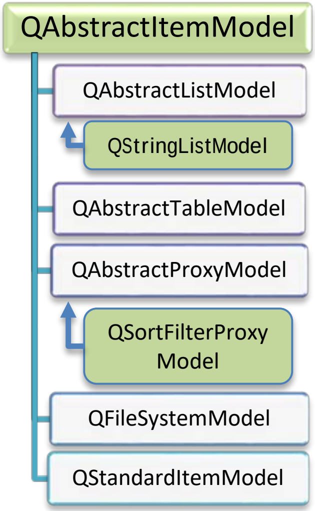
ΣMERTXE

View Classes

- **QtQuick ItemView**
  - Abstract base class for scrollable views
- **QtQuick ListView**
  - Items of data in a list
- **QtQuick GridView**
  - Items of data in a grid
- **QtQuick PathView**
  - Items of data along a specified path

Model Classes

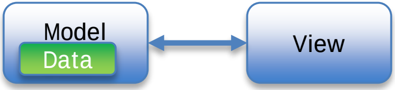
- **QAbstractItemModel**
  - Abstract interface of models
- **Abstract Item Models**
  - Implement to use
- **Ready-Made Models**
  - Convenient to use
- **Proxy Models**
  - Reorder/filter/sort your items
- Model class documentation

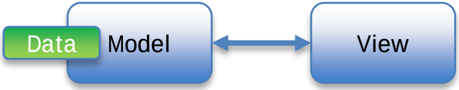


ΣMERTXE

Data-Model-View Relationships

- **Standard Item Model**
  - Data+Model combined
  - View is separated
  - Model is your data
- **Custom Item Models**
  - Model is adapter to data
  - View is separated





QModelIndex

- Refers to item in model
- Contains all information to specify location
- Located in given row and column
  - May have a parent index
- **QModelIndex API**
  - `row()` - row index refers to
  - `column()` - column index refers to
  - `parent()` - parent of index
    - or `QModelIndex()` if no parent
  - `isValid()`
    - Valid index belongs to a model
    - Valid index has non-negative row and column numbers
  - `model()` - the model index refers to
  - `data( role )` - data for given role

Table / Tree

- **Rows and columns**
  - Item location in table model
  - Item has no parent (parent.isValid() == false)

```
indexA = model->index(0, 0, QModelIndex());
indexB = model->index(1, 1, QModelIndex());
indexC = model->index(2, 1, QModelIndex());
```

- **Parents, rows, and columns**
  - Item location in tree model

```
indexA = model->index(0, 0, QModelIndex());
indexC = model->index(2, 1, QModelIndex());
// asking for index with given row, column
and parent
indexB = model->index(1, 0, indexA);
```

Item and
Item Roles

- **Item performs various roles**
  - for other components (delegate, view, …)
- **Supplies different data**
  - for different situations
- **Example:**
  - `Qt::DisplayRole` used displayed string in view
- **Asking for data**

```
QVariant value = model->data(index, role);
// Asking for display text
QString text = model->data(index,
Qt::DisplayRole).toString()
```

- **Standard roles**
  - Defined by Qt::ItemDataRole

Showing simple Data

QStandardItemModel - Convenient Model

- QStandardItemModel
  - Classic item-based approach
  - Only practical for small sets of data

```
model = new QStandardItemModel(parent);
item = new QStandardItem("A (0,0)");
model->appendRow(item);
model->setItem(0, 1, new QStandardItem("B
(0,1)"));
item->appendRow(new QStandardItem("C (0,0)"));
```

Demo

- *"B (0,1)" and "C (0,0)" - Not visible. (list view is only 1-dimensional)*

Proxy Model

- QSortFilterProxyModel
  - Transforms structure of source model
  - Maps indexes to new indexes

```
view = new QQuickView(parent);
// insert proxy model between model and
view
proxy = new
QSortFilterProxyModel(parent);
proxy->setSourceModel(model);
view->engine()->rootContext()-
>setContextProperty("_proxy", proxy);
```

Note: Need to load all data to sort or filter

Sorting/Filtering

- **Filter with Proxy Model**

```
// filter column 1 by "India"
proxy->setFilterWildcard("India");
proxy->setFilterKeyColumn(1);
```

- **Sorting with Proxy Model**

```
// sort column 0 ascending
proxy->sort(0, Qt::AscendingOrder);
```

- **Filter via TextInputs signal**

```
TextInput {
onTextChanged: _proxy.setFilterWildcard(text)
}
```

- Demo

Summary

- **Model Structures**
  - List, Table and Tree
- **Components**
  - Model - Adapter to Data
  - View - Displays Structure
  - Delegate - Paints Item
  - Index - Location in Model
- **Views**
  - ListView
  - GridView
  - PathView

- **Models**
  - QAbstractItemModel
  - Other Abstract Models
  - Ready-Made Models
  - Proxy Models
- **Index**
  - `row(),column(),parent()`
  - `data( role )`
  - `model()`
- **Item Role**
  - Qt::DisplayRole
  - Standard Roles in `Qt::ItemDataRoles`

Model/View

- Model/View Concept
- Custom Models
- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Implementing a Model

- Variety of classes to choose from
  - **QAbstractListModel**
    - One dimensional list
  - **QAbstractTableModel**
    - Two-dimensional tables
  - **QAbstractItemModel**
    - Generic model class
  - **QStringListModel**
    - One-dimensional model
    - Works on string list
  - **QStandardItemModel**
    - Model that stores the data
- **Notice:** Need to subclass *abstract* models

Step 1:
Read Only List Model

```
class MyModel: public QAbstractListModel {
public:
   // return row count for given parent
   int rowCount( const QModelIndex &parent)
   const;
   // return data, based on current index and
   requested role
   QVariant data( const QModelIndex &index,
   int role = Qt::DisplayRole) const;
};
```

Demo

EMERTXE

Step 2:
Header Information

```
QVariant MyModel::headerData(int section,
Qt::Orientation orientation,
int role) const
{
   // return column or row header based on
   orientation
}
```

Demo

Step 3: Enabling Editing

```
// should contain Qt::ItemIsEditable
Qt::ItemFlags MyModel::flags(const QModelIndex &index)
const
{
    return QAbstractListModel::flags() |
    Qt::ItemIsEditable;
}// set role data for item at index to value
bool MyModel::setData( const QModelIndex & index,
const QVariant & value,
int role = Qt::EditRole)
{
    ... = value; // set data to your backend
    emit dataChanged(topLeft, bottomRight); // if
    successful
}
```

Demo

Step 4:
Row Manipulation

```
// insert count rows into model before row
bool MyModel::insertRows(int row, int count, parent)
{
   beginInsertRows(parent, first, last);
   // insert data into your backend
   endInsertRows();
}
// removes count rows from parent starting with row
bool MyModel::removeRows(int row, int count, parent)
{
   beginRemoveRows(parent, first, last);
   // remove data from your backend
   endRemoveRows();
}
```

Demo

Hands-on

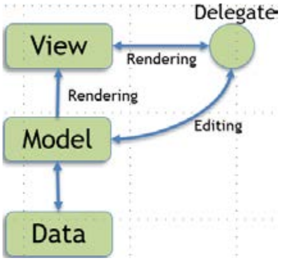
- Lab 11: City list model
  - Objectives
  - Template code

Model/View

- Model/View Concept
- Custom Models
- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Item Delegates

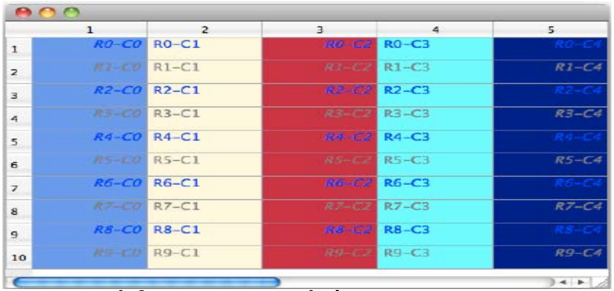
- QAbstractItemDelegate subclasses
  - Control appearance of items in views
  - Provide edit and display mechanisms
- QItemDelegate, QStyledItemDelegate
  - Default delegates
  - Suitable in most cases
  - Model needs to provide appropriate data
- When to go for Custom Delegates?
  - More control over appearance of items



Item Appearance

Data table

shown has no custom delegate

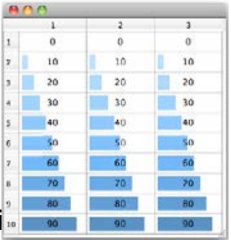


	1	2	3	4	5
1	R0-C0	R0-C1	R0-C2	R0-C3	R0-C4
2	R1-C0	R1-C1	R1-C2	R1-C3	R1-C4
3	R2-C0	R2-C1	R2-C2	R2-C3	R2-C4
4	R3-C0	R3-C1	R3-C2	R3-C3	R3-C4
5	R4-C0	R4-C1	R4-C2	R4-C3	R4-C4
6	R5-C0	R5-C1	R5-C2	R5-C3	R5-C4
7	R6-C0	R6-C1	R6-C2	R6-C3	R6-C4
8	R7-C0	R7-C1	R7-C2	R7-C3	R7-C4
9	R8-C0	R8-C1	R8-C2	R8-C3	R8-C4
10	R9-C0	R9-C1	R9-C2	R9-C3	R9-C4

- No need for custom delegate!
- Use `Qt::ItemRole` to customize appearance

QAbstractItemDelegate

```
class BarGraphDelegate : public
QAbstractItemDelegate {
public:
   void paint(QPainter *painter,
   const QStyleOptionViewItem &option,
   const QModelIndex &index) const;
   QSize sizeHint(const QStyleOptionVi
&option,
   const QModelIndex &index) const;
};
```



	1	2	3
1	0	0	0
2	10	10	10
3	20	20	20
4	30	30	30
5	40	40	40
6	50	50	50
7	60	60	60
8	70	70	70
9	80	80	80
10	90	90	90

[Demo]

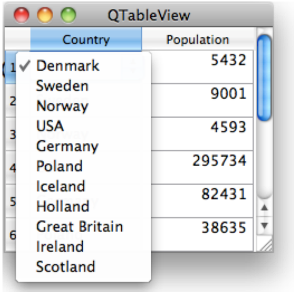
[Documentation]

Model/View

- Model/View Concept
- Custom Models
- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Editor Delegate

- Provides QComboBox
  - for editing a series of values



```
class CountryDelegate : public QItemDelegate
{
    public:
    // returns editor for editing data
    QWidget *createEditor( parent, option, index ) const;
    // sets data from model to editor
    void setEditorData( editor, index ) const;
    // sets data from editor to model
    void setModelData( editor, model, index ) const;
    // updates geometry of editor for index
    void updateEditorGeometry( editor, option, index ) const;
};
```

Creating Editor

- Create editor by index

```
QWidget *CountryDelegate::createEditor( ... ) const {
  QComboBox *editor = new QComboBox(parent);
  editor->addItems( m_countries );
  return editor;
}
```

- Set data to editor

```
void CountryDelegate::setEditorData( ... ) const {
  QComboBox* combo = static_cast<QComboBox*>( editor );
  QString country = index.data().toString();
  int idx = m_countries.indexOf( country );
  combo->setCurrentIndex( idx );
}
```

Data to the model

- When user finished editing
  - view asks delegate to store data into model

```
void CountryDelegate::setModelData(editor, model,
index) const {
QComboBox* combo = static_cast<QComboBox*>(
editor );
model->setData( index, combo->currentText() );
}
```

- If editor has finished editing

```
// copy edtitors data to model
emit commitData( editor );
// close/destroy editor
emit closeEditor( editor, hint );
// hint: indicates action performed next to
editing
```

Editor's geometry

- Delegate manages editor's geometry
- View provides geometry information
  - QStyleOptionViewItem

```
void CountryDelegate::updateEditorGeometry( ... ) const
{
  // don't allow to get smaller than editors sizeHint()
  QSize size = option.rect.size().expandedTo(editor->
sizeHint());
  QRect rect(QPoint(0,0), size);
  rect.moveCenter(option.rect.center());
  editor->setGeometry( rect );
}
```

- Demo
- Case of multi-index editor
  - Position editor in relation to indexes

Setting Delegates

- view->setItemDelegate(…)
- view->setItemDelegateForColumn(…)
- view->setItemDelegateForRow(…)

Type Based
Delegates

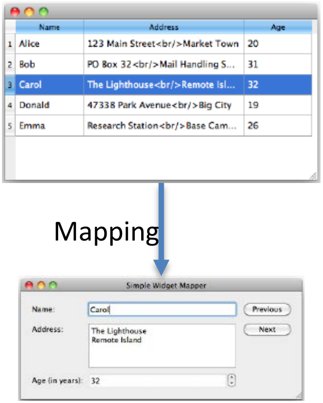
Demo

Model/View

- Model/View Concept
- Custom Models
- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

QDataWidgetMapper

- Maps model sections to widgets
- Widgets updated, when current index changes
- Orientation
  - Horizontal => Data Columns
  - Vertical => Data Rows



	Name	Address	Age
1	Alice	123 Main Street<br/>Market Town	20
2	Bob	PO Box 32<br/>Mail Handling S...	31
3	Carol	The Lighthouse<br/>Remote Isl...	32
4	Donald	47338 Park Avenue<br/>Big City	19
5	Emma	Research Station<br/>Base Cam...	26

Name: Carol

Address: The Lighthouse
Remote Island

Age (in years): 32

QDataWidgetMapper

- Mapping Setup

```
mapper = new QDataWidgetMapper(this);
mapper->setOrientation(Qt::Horizontal);
mapper->setModel(model);
// mapper->addMapping( widget, model-section)
mapper->addMapping(nameEdit, 0);
mapper->addMapping(addressEdit, 1);
mapper->addMapping(ageSpinBox, 2);
// populate widgets with 1st row
mapper->toFirst();
```

- Track Navigation

```
connect(nextButton, SIGNAL(clicked()),
        mapper, SLOT(toNext()));
connect(previousButton, SIGNAL(clicked()),
        mapper, SLOT(toPrevious()));
```

Demo

Mapped Property

```
class QLineEdit : public QWidget
{
   Q_PROPERTY(QString text
   READ text WRITE setText NOTIFY textChanged
   USER true) // USER property
};
```

- USER indicates property is user-editable property
- Only one USER property per class
- Used to transfer data between the model and the widget

```
addMapping(lineEdit, 0); // uses "text" user property
addMapping(lineEdit, 0, "inputMask"); // uses named property
```

Demo

Model/View

- Model/View Concept
- Custom Models
- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

Drag and Drop for Views

- Enable the View

```
// enable item dragging
view->setDragEnabled(true);
// allow to drop internal or external items
view->setAcceptDrops(true);
// show where dragged item will be dropped
view->setDropIndicatorShown(true);
```

- Model has to provide support for drag and drop operations

```
Qt::DropActions MyModel::supportedDropActions() const
{
    return Qt::CopyAction | Qt::MoveAction;
}
```

- Model needs to support actions
  - For example `Qt::MoveAction`
  - implement `MyModel::removeRows( ... )`

QStandardItemModel

- Setup of Model
  - Model is ready by default
  - model->mimeTypes()
    - "application/x-qabstractitemmodeldatalist“
    - "application/x-qstandarditemmodeldatalist"
  - model->supportedDragActions()
    - `QDropEvent::Copy | QDropEvent::Move`
  - model->supportedDropActions()
    - `QDropEvent::Copy | QDropEvent::Move`
- Setup of Item

```
item = new QStandardItem("Drag and Droppable Item");
// drag by default copies item
item->setDragEnabled(true);
// drop mean adding dragged item as child
item->setDropEnabled(true);
```

Demo

QAbstractItemModel

```
class MyModel : public QAbstractItemModel {
public:
    // actions supported by the data in this model
    Qt::DropActions supportedDropActions() const;
    // for supported index return  Qt::ItemIs(Drag|Drop)Enabled
    Qt::ItemFlags flags(const QModelIndex &index) const;
    // returns list of MIME types that are supported
    QStringList QAbstractItemModel::mimeTypes() const;
    // returns object with serialized data in mime formats
    QMimeData *mimeData(const QModelIndexList &indexes) const;
    // true if data and action can be handled, otherwise false
    bool dropMimeData(const QMimeData *data, Qt::DropAction
    action,
    int row, int column, const QModelIndex &parent);
};
```

Demo

Model/View

- Model/View Concept
- Custom Models
- Delegates
- Editing item data
- Data Widget Mapper
- Drag and Drop
- Custom Tree Model

A Custom Tree Model in 5 Steps

1. Read-OnlyModel
2. EditableModel
3. Insert-RemoveModel
4. LazyModel
5. Drag and DropModel

A Node Structure

```
class Node {
public:
   Node(const QString& aText="No Data",   Node
*aParent=0);
   ~Node();
   QVariant data() const;
public:
   QString text;
   Node *parent;
   QList<Node*> children;
};
```

Demo (node.h)

Read-Only Model

```
class ReadOnlyModel : public QAbstractItemModel {
public:
   ...
   QModelIndex index( row, column, parent ) const;
   QModelIndex parent child ) const;
   int rowCount( parent ) const;
   int columnCount( parent ) const;
   QVariant data( index, role) const;
protected: // important helper methods
   QModelIndex indexForNode(Node *node) const;
   Node* nodeForIndex(const QModelIndex &index)
   const;
   int rowForNode(Node *node) const;
};
```

Editable Model

```
class EditableModel : public ReadOnlyModel {
public:
    …
    bool setData( index, value, role );
    Qt::ItemFlags flags( index ) const;
};
```

Insert / Remove Model

```
class InsertRemoveModel : public
EditableModel {
public:
    ...
    void insertNode(Node *parentNode, int
    pos, Node *node);
    void removeNode(Node *node);
    void removeAllNodes();
};
```

Lazy Model

```
class LazyModel : public ReadOnlyModel {
public:
   ...
   bool hasChildren( parent ) const;
   bool canFetchMore( parent ) const;
   void fetchMore( parent );
};
```

DnD Model

```
class DndModel : public InsertRemoveModel {
public:
   ...
   Qt::ItemFlags flags( index ) const;
   Qt::DropActions supportedDragActions() const;
   Qt::DropActions supportedDropActions() const;
   QStringList mimeTypes() const;
   QMimeData *mimeData( indexes ) const;
   bool dropMimeData(data, dropAction, row, column,
   parent);
   bool removeRows(row, count, parent);
   bool insertRows(row, count, parent);
};
```

QtMultimedia

QtMultimedia

- Qt Multimedia is an essential module that provides a rich set of QML types and C++ classes to handle multimedia content.
- It also provides necessary APIs to access the camera and radio functionality.

Features

- Access raw audio devices for input and output
- Play low latency sound effects
- Play media files in playlists (such as compressed audio or video files)
- Record audio and compress it
- Tune and listen to radio stations
- Use a camera, including viewfinder, image capture, and movie recording
- Play 3D positional audio with Qt Audio Engine
- Decode audio media files into memory for processing
- Accessing video frames or audio buffers as they are played or recorded

Audio

- Qt Multimedia offers a range of audio classes, covering both low and high level approaches to audio input, output and processing.
- For playing media or audio files that are not simple, uncompressed audio, you can use the QMediaPlayer C++ class.
- The QMediaPlayer class and associated QML types are also capable of playing video, if required.
- The compressed audio formats supported does depend on the operating system environment, and also what media plugins the user may have installed.
- For recording audio to a file, the QAudioRecorder class allows you to compress audio data from an input device and record it.
- Demo

EMERTXE

Video

- We can use the QMediaPlayer class to decode a video file, and display it using QVideoWidget, QGraphicsVideoItem, or a custom class.
- Demo

THANK YOU